A Guide To Your

RAOS

Including:

- DESKTOP
- SCREENS 2.0

(version 1.0 and enhancements.)

Zobian - Controls

RAOS

RAT ACTUATED OPERATING SYSTEM

ZOBIAN CONTROLS

DESKTOP USER MANUAL

Introduction

Welcome to the world of RAOS. This is a system which provides the efficiency and convenience found on some of the most powerful computers available today. The Desktop is a program which functionally replaces the Disk Utility Program (DUP.SYS) from Atari. With it, you can perform most disk maintenance easier than you can imagine. The Zobian SuperRat interfaces with the Desktop to allow you the ability to point at files to perform action on. The only instance in which you will be required to type any keyboard input, is when you rename a file.

Since it is a replacement for DUP, the Desktop is accessed in the same manner. Simply by typing DOS (from BASIC), you can enter the Desktop, perform your required functions, and return to BASIC. Two disk drive systems are supported to enable fast copying.

We, at Zobian Controls, feel that once you begin to use the SuperRat and the Desktop, you'll wonder how you managed without them.

Setting Up

Before using any of the programs on the Desktop master disk, be sure to backup the disk, and store the original in a safe place. From this point on, use the backup. Follow this procedure to enter the Desktop:

1. Turn on your disk drive.
2. Open the drive door and put the Desktop disk backup into it.
3. Make sure no cartridges are inserted into the computer. If you have an 800XL, 600XL, 1200XL, 65XE, or 130XE, hold down the OPTION button.
4. Turn on the computer. If you are using one of the machines listed above, you may release the OPTION key once you hear the data on the disk being loaded. This requires turning up the volume a bit.

5. Once the disk stops loading, you will be presented with the Desktop screen.
6. Make sure your SuperRat is plugged into Joystick port 1.
7. At this time, you should be able to move the cursor (a cross) around the Desktop. If not, follow the directions in the debugging section of the manual.
8. If everything seems okay, then welcome to the Desktop!

USING THE DESKTOP

The center portion of the Desktop contains a large box which shows all of the files on the current disk. Since we just booted with the Desktop disk, verify that everything is there by comparing your display with the following list.

DOS.SYS.	this is a modified version of Atari's DOS.
SYNTEL.SYS.	the Desktop file itself.
AUTORUN.SYS.	the SuperRat driver program.
ACCU.RAT.	a drawing program using the SuperRat.
AUTORAT.BAS.	a basic program which demonstrates how to write your own programs using the SuperRat.

The empty rectangle below the file box is the message box. This is where the Desktop and you communicate. If any errors occur, they will be detailed here. In addition, when you choose to rename a file, you will enter the new filename here.

Before we examine the functions available on the Desktop, take this opportunity to get the feel of the SuperRat. Files can be turned on and off. This is done by moving the cursor over a filename and pressing the left button. It should be mentioned that all actual selections are made with the left button. When the button is pressed in this manner on a file, it lights up in inverse video to show that it is selected. You can do this to as many files as you like. To de-select, simply press the button again when the cursor is over the file in question.

To the left of the file box are five function boxes relating to file utilities. These are discussed in order from top to bottom. Do not use any of these functions until you understand what they do and how they work. Doing so could damage the data on the diskette in the drive. Notice that if you select any of these functions when a file is not selected, nothing will happen.

DELETE: When this function is selected, any files selected in the file box are targeted for elimination from the disk. Before such a terrible thing happens however, a message reading "PLEASE CONFIRM" appears in the message box. To confirm, move the cursor into the Delete box and press the button. To abort, move the cursor away from the Delete box and press the button. Once confirmed, all files selected are deleted and the file box is updated with the new directory.

RENAME: Use this function to change the name of the file which is highlighted. If more than one file is selected, the first filename will be changed. Once you press RENAME, a 'D:' will appear in the message box. Type in the new name, press return and the new directory will be displayed indicating the change.

PROTCT: This button allows you to lock (protect) or unlock any file which is selected. If the file is currently unlocked, it will be locked and vice verse. This works in combinations as well.

COPY: The use of this function represents one of the most powerful capabilities of the Desktop. Any files highlighted in the file box will be copied to another disk or another drive by simply pressing a few buttons. The same operation using DUP.SYS would be far more time consuming, require many keystrokes, and be error prone. Once selected, the message box displays 'DESTINATION?'. This is a prompt in which the Desktop is determining how many drives are involved. If you are fortunate enough to own two drives, move the cursor to the box below the message box and on top of the box containing the letter 'B' and press the button. Otherwise, move the cursor to the letter 'A' and select. Please note that for ALL copies, the source drive is drive zero or 'A', and the destination (for two drive systems) is drive 1 or 'B'. In a two drive system, you need do no more. The copying will take place. With a single drive system, you will see a prompt in the message box as such: 'INSERT SOURCE'. The source disk should already be in the drive. Just press the button and the first file will be read in. Next, a prompt reading 'INSERT TARGET' will appear. Remove the source disk and place the disk to be copied to in the drive and press the button. The file just read in will be written to it. This process is repeated until all files on the source disk which were selected, are copied. At this point, you will see the directory of the target disk showing that all of the files were indeed copied. If an error develops, the entire process is aborted.

LOAD: When this function is selected, the highlighted file is loaded (if possible). The Desktop temporarily disengages to allow a loaded file to execute, if necessary. If the file loads and returns, the Desktop will re-execute as if nothing has happened. The modified DOS.SYS is required for this reason.

On the right side of the file box are three more function boxes which perform action on a non-file basis. They do not require that a file be selected, and will ignore any that are. They are discussed in detail following:

FRMT: This function formats or initializes a disk. This process will erase all information on the disk in the drive. When selected, the message box displays the prompt 'PLEASE CONFIRM'. To proceed with the format process, move the cursor to the FRMT box again and press the button. To abort, move the cursor away from the FRMT box, and press the button. Please note that all formatting is done on drive 'A'.

VRFY: This handy function will toggle the current write verify mode. If the computer is set to verify writes, it will be set to write without verification, and the current (new) mode will be indicated in the message box.

WDOS: After a disk has been formatted, it cannot be used to boot the computer unless DOS.SYS exists on it. DOS.SYS cannot be copied as a file since it requires that three boot sectors be written to the disk to start it. Selecting this function will instruct you to "INSERT TARGET' and press the SuperRat button. The modified DOS.SYS will be written to the disk in the drive and you will now be able to boot the new disk. To make it a complete SuperRat disk, you should copy the file SYNTEL.SYS to it also.

The only function boxes not discussed as yet, are those grouped below the message box. We will explain their use now:

A and B: These buttons were referred to before as the destination selectors for the copy function. They also have functionality by themselves. By selecting either, you instruct the Desktop to retrieve and display the directory for the appropriate drive.,

EXIT: This is the normal exit from the Desktop. If called from BASIC, you will be returned there. Otherwise, the result will probably be a reboot, in which case you will end up back in the Desktop. You can accomplish the same thing by pressing the RESET key.

UP and DOWN arrows: If there are more files in the directory than can be displayed in the file box, these function boxes allow you to page through them.

DEBUGGING THE DESKTOP

If you have problems bringing up the Desktop or using it once it is running, try checking the following symptoms:

Is the proper DOS.SYS on the disk and did you boot with it??
— if you are in doubt on this, boot a direct copy of the original master disk and select WDOS to put a new copy on the disk you are trying to work with.

Is the file SYNTEL.SYS on the disk?
— don't rename this file or a DOS call won't recognize it.

Is your SuperRat plugged into joystick port one (1)?
— it won't work anywhere else!

DESKTOP CREDITS

Desktop was designed by Bob Dolan and programmed by Brian Ernisse of Syntel.

The Desktop manual was written by Bob Dolan.

SCREENS 2.0

This portion of the booklet describes SCREENS 2.0, the windowing environment for RAOS. It is an enhanced version of the highly-acclaimed SCREENS 1.0, incorporating the SuperRAT for window mobility. The following is from the original SCREENS 1.0 booklet, with several pages at the end detailing the enhancements that make it SuperRAT compatible. The result, SCREENS 2.0, gives the Atari 8-bit user movable windows that can be changed in size. It is the most advanced windowing environment for the Atari 8-bit computers, and will be supported by many new software titles that come from other software developers.

Included on your RAOS diskette is a demonstration program called "DEMO.BAS." It **cannot** be loaded from DESKTOP. To view, simply boot up your RAOS disk and type RUN "D:DEMO.BAS". It'll give you a good idea of the awesome windowing power your computer is capable of.

Additional Specifications — SCREENS 2.0

Window Types

SCREENS 2.0 provides enhanced window capability. These more powerful windows can be controlled much like GEM windows. In addition, built in RAT monitoring and pointer control are provided. These functions are automatic and require no user or programmer interaction.

Several new XIO command codes are available;

103 — RESET WINDOW PRIORITY TABLE. This is used to reset SCREENS internal status. It should be used in any initialization procedure.
ex. XIO 103, #1, 0, 0, "W:"

104 — SET SCREENS 2.0 WORKSPACE BUFFER. Some of SCREENS functions require the use of extra memory for temporary buffers.
ex. XIO 104, #1, BUFFLO, BUFFHI, "W:"

105 — OPEN WINDOW TO DISPLAY. Actually places a window with attributes on the screen. The buffer location may also be specified to allow SCREENS to update the window's contents automatically when manipulated.
ex. XIO 105, #1, WBUFFLO, WBUFFHI, "W:"

106 — CLOSE WINDOW ON DISPLAY. Removes and erases the addressed window.
ex. XIO 106, #1, 0, 0, "W:"

107 — MAKE WINDOW ACTIVE. If a window (which is already opened) is not the active one, this command will make it so. Invoking it will place attributes on the addressed window and remove them from the window previously active.
ex. XIO 107, #1, 0, 0, "W:"

108 — MOVE WINDOW. Once this call is made, SCREENS will read the rat and move an outline of the active window with it as long as the rat's button is depressed. At this time the window's previous position will be restored, and the window will be redrawn at the new position.
ex. XIO 108, #1, 0, 0, "W:"

109 — RESIZE WINDOW. Functions in the same manner as a window move but in the sense of altering the window's size.
ex. XIO 109, #1, 0, 0, "W:"

110 — MAKE WINDOW FULL SCREEN. Causes the window to take up the full screen. If the windows buffer, specified when it was opened, is not large enough to hold the data, data may be lost upon subsequent resizing.
ex. XIO 110, #1, 0, 0, "W:"

SCREENS 2.0 Memory Locations

There is a table of user accessable memory locations in SCREENS version 2. Some of these are functions and must be called with a USR call (in BASIC). The others are status registers which must be PEEKed and POKEd.

FUNCTIONS

BASRR=12085 Resets the RAT control.

BASRON=12125 Turns the RAT monitoring on.

BASROF=12141 Turns off the background monitoring of the RAT. This should be done during the critical operations such as disk I/O.

BASRP=12324 This calls a routine which will update the RATLOC table. Refer to that section for details.

BASGR=12551 Calls a routine which updates the RATs position relative to the graphics mode. (GRATX & GRATY)

BASCR=11853 Rests the pointer or cursor's control.

BASCON=11871 Turns on SCREEN's control of the RAT pointer. This is also a background routine internal to SCREENS 2.

BASCOF=11905 Stops RAT pointer updates.

All of these functions can be called from BASIC in the following manner: A=USR(BASGR). Note that none of the functions require that any parameters are passed and nothing is returned in the variable A.

Memory Locations

RATX(LO)	=13196
RATX(HI)	=13197
RATY	=13198

These are the RAT position registers. They will be constantly updated with the x and y location of the RAT if the interrupt routine is enabled.

ex. X=PEEK (13197) *256+PEEK (13196) : Y=PEEK (13198)

Note that x will always range from 0 to 319 and y from 0-179.

GRATX (LO)	=13199
GRATX (HI)	=13200
GRATY	=13201

When writing programs in graphics modes other than GRAPHICS 8, these registers will be most useful. The values found here will rely on which mode the screen is set to.

WRATX(LO)	=13203
WRATX (HI)	=13204
WRATY	=13205

These registers are used to find the RAT's location relative to the upper left hand corner of the active window.

CIMAGE=12796 to 12804

This block of memory holds the image of the pointer which will be displayed on the screen. It is set to a default of an arrow when the system is booted. Changing the image is accomplished by simply POKEing this block with the binary values corresponding to the image you prefer. The layout is identical to a player.

RATLOC=13202

This is a two part register. The value returned on a PEEK contains the window number that the RAT is over in the high nibble, and the specific window region code in the lower nibble. To derive these values, use the following BASIC code:

```
WIND=INT (PEEK (13202)/16): REGION=PEERK (13203) - WIND*16
```

RATLOC Region Codes

0	Off active window	3	Full screen box	6	Down arrow	9	Bottom slider
1	Close box	4	Up arrow	7	Re-size box	10	Left arrow
2	Mover bar	5	Right slider	8	Right arrow	11	Window interior

Table of contents

Overview

Each of the multiplicity of standard display modes on ATARI computers consist of columns and rows of pixels; pixels are the smallest discretely addressable picture elements in the display. The number of columns and rows and the size and nature of the pixels vary from mode to mode. For example, in GRAPHICS 24 the display consists of 320 columns and 192 rows of fine pixels which can either be on or off. In GRAPHICS 0, the display consists of 40 columns and 24 rows of pixels which actually comprise characters for this display mode.

One consistency across all the ATARI display modes is the co-ordinate system used to locate a pixel on the display. The ATARI system designates the leftmost pixel column of the display as column 0 and the topmost pixel row as row 0. Pixel positions are then referenced to this origin. Thus, a pixel which is in the tenth column from the left and the fifth row from the top is in column 9, row 4.

What "SCREENS" allows you to do is to designate an arbitrary rectangular area in a display as an independent screen, or window, and to output to or input from that window without effecting the rest of the display. The way you designate the area of the display the window is to occupy is to specify (1) the column and row pixel position in standard co-ordinates of the window's upper left corner and (2) the number of columns and rows the window is to contain.

Position and size are just two of the characteristics you are able to specify for each window. We'll discuss the others later. But it brings up the question of how you communicate such information to the system. The ATARI operating system supports such communication through its Central Input Output (CIO) protocol. To take advantage of this protocol, "SCREENS" installs a device called "W:". This device handles all the functions supported by "SCREENS" including defining a window, writing to it, reading from it and all the special functions to be described later.

ATARI BASIC provides a number of commands which allow the programmer to interface with CIO. These include OPEN, CLOSE, GET, PUT, INPUT, PRINT, ENTER, LIST, NOTE, POINT, STATUS and XIO. Each of these functions can be used with device "W:". In the discussion which follows, these BASIC commands will be used in examples which demonstrate "SCREENS" features, as it is assumed that the reader has familiarity with ATARI BASIC. Of course, any language which uses the CIO protocol can communicate with "W:" in an analogous fashion. The details of this communication are discussed in Section 3.

Loading the program

Turn on your disk drive. When the busy light extinguishes, insert the "SCREENS" program diskette in the drive (label up and towards you) and close the door. Turn the computer power switch on (non-XL models should have the BASIC cartridge installed). The ATARI disk operating system (DOS) and "SCREENS" will both autoload. When the autoload is complete, the program name will be displayed along with the version number; below it the copyright notice and the READY prompt will appear.

"SCREENS" supports the use of the ATARI 850 Interface; if you wish the "R:" handler to be installed at power-up, turn on the 850 Interface before you power up your computer. It will auto-load along with DOS and "SCREENS".

"SCREENS" exists on your program diskette as a file named AUTORUN.SYS. Neither the diskette nor the file are copy-protected. To transfer the AUTORUN.SYS file to another diskette, follow the DOS instructions for file copying. On any ATARI DOS diskette to which the "SCREENS" AUTORUN.SYS file is copied, it is a wise precaution to also create a MEM.SAV file. Without it, a DOS call from BASIC will corrupt the "W:" device handler.

Some DOSs do not support the immediate loading and execution of AUTORUN.SYS files on power-up. For these DOSs, follow the instructions for loading machine language files.

Initial conditions

Once "SCREENS" has auto-loaded, the "W:" device handler is installed. It will neither interfere with normal system operation nor be lost on SYSTEM RESET. The only noticeable system difference (other than the existence of "W:") is a 3316 byte decrease in free RAM. It is in this "stolen" RAM that the "W:" device handler resides.

Opening a window

Before you can use a window, it must first be opened. This is done, logically enough, using the OPEN statement. The format of the OPEN statement for device "W:" is

```
OPEN #iocb_no,io_type,reset_flag,device_id
```

where iocb_no is an integer from 1 to 8 which specifies the input-output control block used to communicate with the device being opened, io_type is an integer which specifies the direction of data flow (4=input, 8=output and 12=both), reset_flag is an integer which, if non-zero, causes the parameters of the opened device to be reset and device_id is a string constant or variable of the form "W#:" where # specifies a unit number from 1 through 9 ("W:" defaults to "W1:").

That's a lot of words, which may become easier to understand if we use an example. Let's interpret the OPEN statement:

```
OPEN #3,8,0,"W6:"
```

What it says is 'assign unit 6 of device "W:" to IOCB 3 for output and don't reset that unit's current parameters'.

If we wanted the parameters reset, we could change the statement to read

```
OPEN #3,8,1,"W6:"
```

If we further wanted to OPEN the device for both input and output, we would use

```
OPEN #3,12,1,"W6:"
```

We could also use variables in the statement. For example,

```
IOCB=3:IOTYPE=12:RESET=1:A$="W6:":OPEN #IOCB,IOTYPE,RESET,A$
```

is equivalent to the OPEN statement in the previous paragraph which used only constants.

To close a window, one issues the CLOSE command. The format for this command is simply

```
CLOSE #iocb_no
```

where iocb_no is the same number used for opening the window. The result of the CLOSE command is to close off the previously opened CIO channel for communication to the window and free up that channel for other use. Neither the display nor the window state is altered by the CLOSE command.

As noted above, device "W:" supports units one through nine. This means that up to nine independent windows can be defined at any one time. Each will have its own set of parameters. The windows can be defined to overlap on the display. If they do so, then what happens in one may effect what's displayed in the other. If you wish to overlay windows in your program, you have to be mindful of possible interaction. "SCREENS" provides a way for you to save individual windows and recall them back to the display. We'll describe that process later.

Window parameters

Right now let's discuss the parameters which the user gets to set for each window. There are fifteen altogether. Four have to do with window placement. Two determine window position; the user sets the pixel position (column and row) of the upper left corner of the window. Two determine window size; the user sets width and height of the window in pixels.

Seven parameters deal with the display of text in the window. Two of these determine character placement; the user sets the row and column offset in pixels from the upper left corner of the window to the upper left corner of the character cell of the next character to be displayed. These two parameters allow you to position text randomly inside the window. Two parameters determine character size; the user sets the height and width of the character cell in pixels. The character cell is the size of the box on the display set aside for each character.

The three other parameters having to do with text display are as follows. One parameter is the base address of the character set to be used. This parameter can be changed to take advantage of user-defined character sets. One parameter is a flag which tells the device whether the selected character set is defined as an 8 by 8 matrix (like the standard ROM character set) or whether it is a specially designed 16 by 16 higher resolution set ("SOFT.SET", an example of a 16 by 16 font, is on your "SCREENS" diskette). One parameter sets the color to be used to display the character; it is analogous to the BASIC COLOR parameter but applies only to text printed in the window.

The four remaining parameters do a variety of useful things. One parameter is the ASCII value of the character to be used as the cursor when receiving input from a window; the cursor can also be turned off. One parameter sets the logic to be used when displaying a character; you have the choice of AND, EOR, OR or overwrite. This effects the way a character is printed atop information which may already exist on the display. One parameter locks the window to prevent it from clearing or scrolling. Finally, one parameter is the value of np_index; the use of this parameter will be discussed later.

When the user specifies that the window parameters be reset when a window is opened, the parameters take on the following values. The upper left corner of the window is set to the upper left corner of the display. The window size is set to the maximum window size (320 by 192). The character cell is placed in the upper left corner of the window and sized to 8 by 8. The base address of the character set is set to that of the ROM set, and the character size flag is zeroed indicating an 8 by 8 matrix. The character color is set to 255, which is interpreted as COLOR 1 in a two-color mode, COLOR 3 in a four color mode, and so on. The input cursor is an inverse space, the display logic is overwrite, the window is unlocked and np_index is zero.

Printing to a window

Let's write a little program to see what all of this means. We'll use graphics mode 8 as our display mode and write "HELLO" to the graphics display.

```
 10 GRAPHICS 8
 20 OPEN #1,8,1,"W:"
 80 PRINT #1;"HELLO"
150 CLOSE #1
160 END
```

If you run this program with "SCREENS" installed, you will find that, as advertised, the word "HELLO" shows up in the upper left hand corner of the display. Note that the text looks just like standard ATARI text because we are using the ROM character set and printing into an 8 by 8 character cell, the same as is used in the GRAPHICS 0 text mode.

You should have expected the "H" to show up where it did on the display. As was stated, when the parameters are reset, the window and the character cell are placed in the upper left corner of the display. But what about the rest of "HELLO"? Note that each character is positioned one character cell width to the right of the preceding character automatically. That is, the parameters which specify character placement are automatically adjusted after each character is displayed to point to the next character cell to be filled.

Add two lines to our sample program by typing

```
60 FOR X=1 TO 2
90 NEXT X
```

pressing RETURN after each line. Now run the program again. Note that the second "HELLO" prints right below the first. "W:" treats the end-of-line character generated at the end of the first PRINT statement just like the text editor does in GRAPHICS 0; it positions the next character at the beginning of the next line on the display. The only other character which has a special effect on the window is the clear display character, ASCII 125. If the window is not locked, sending this character to a window will cause the window to clear and the character position to be set to the upper left corner of the window.

Add another line to our sample program by typing

```
70 PUT #1,125
```

and pressing RETURN. Now run the program again. Note that the window cleared between printing the first and second "HELLO" and that the second "HELLO" printed at the top left corner of the display.

Now type LIST "W:" and press RETURN. Note that the program is listed on the display right below the "HELLO". Why does this happen? Well, when BASIC lists a program to a device, it first opens it. This process does not reset "W:". Hence the "W:" device handler remembers where the next character was to be placed and continues on from there.

List the program to "W:" a few more times. What happens when the text reaches the bottom of the graphics display? Scrolling! Logic is built in to "W:" which scrolls the text in the window whenever you attempt to print beyond the bottom of the window. As noted above, this can be overridden by locking the window in case scrolling is not desired.

Now let's modify our program a bit. Type

```
10 GRAPHICS 7
```

and press RETURN. Now run the program again. Note that the character cell size is still 8 by 8 and the characters themselves are still from the standard ATARI set, but now they are bigger and in color. The difference is due to the difference in pixel size and type between graphic modes 7 and 8.

List the program to "W:" twice and watch the window scroll. Note that again it does so when the text reaches the bottom of the graphics display. Because of the coarser resolution of graphics mode 7, however, less lines are printed before scrolling must occur. But "W:" knew that and scrolled when it had to! That's because "W:" checks its size and position relative to the current graphics mode every time a character is printed. If the graphics mode is changed between characters such that the boundaries of the window now exceed the boundaries of the display, the window is re-sized automatically. Thus, although we started out by resetting the window size to 320 columns by 192 rows, just before the first character was displayed (the "H" from "HELLO) the window was re-sized to fit in graphics mode 7, i.e. to 160 columns by 80 rows.

What happens now if we switch back to GRAPHICS 8? Let's find out. Type

```
                    GRAPHICS 8
```

and press RETURN. Now list the program to "W:" a few times. Note that "W:" retains the smaller window size that it was forced into while printing to the graphics 7 display.

One further feature of the text output function of "W:" is that it responds to the control-1 protocol, i.e. text display is interrupted if the "1" key is pressed while the CONTROL key is depressed and then continues when this process is repeated. List the program to "W:" again and try it out.

Inputting from a window

The input function supported by the "W:" device handler employs the keyboard handler to actually get characters from the keyboard and then displays them using the window's current character color, size, font and display logic. When input is done through a window, all characters entered at the keyboard (with a few exceptions) are accepted as input, printed to the window and returned to the calling program. The following program is a simple example of inputting through a window. Clear the previous program, then type this program in and run it.

```
 10 GRAPHICS 7
 30 OPEN #1,12,1,"W:"
 80 GET #1,X:? CHR$(27);CHR$(X);:IF X<>155 THEN 80
 90 CLOSE #1
140 END
```

In this example, the GET command is used to input one character at a time. The returned character is then printed in the text window as a check that the input function is working properly. Type a number of different characters from the keyboard to see how the program functions. Pressing the RETURN key, the BREAK key or control-3 ends the program.

There are two keys which act as editing keys during input. The "delete back space" key backs up the displayed cursor one character cell and, if the window is using overwrite display logic, erases the previous character. The returned ASCII value for this key is 126. The "clear" key, when pressed while SHIFT or CONTROL is depressed, moves the cursor to the beginning of the line and, if the window is using overwrite display logic, clears the current input line. The returned ASCII value for these key combinations is 125. Try running the sample program again and note the behavior of these editing keys.

Now let's add some lines to our sample program to demonstrate some additional input features. The following two lines will limit the size of our input window to 128 pixels across by 8 pixels high. Since we are using the default 8 by 8 character cell size, this will limit our window to one line of 16 characters. Type the following

```
40 XIO 100,#1,1,0,"W:"
50 X=128:Y=8:POINT #1,X,Y
```

pressing RETURN after each line.

The next two lines will change our input cursor to the underline character. Type the following

```
                    60 XIO 100,#1,6,0,"W:"
                    70 X=ASC("_"):Y=0:POINT #1,X,Y
```

pressing RETURN after each line.

Now let's set up to enter a string from BASIC. Type the following

```
                    20 DIM A$(15)
                    80 INPUT #1,A$
```

pressing RETURN after each line.

Finally, let's print out the returned string in the text window after inputting it from "W:". Type the following

```
                   100 FOR I=1 TO LEN(A$)
                   110 PRINT CHR$(27);A$(I,I);
                   120 NEXT I
                   130 PRINT
```

pressing RETURN after each line.

Now run the program and experiment with entering data and using the editing keys. Note that if fifteen or fewer characters are entered and then the RETURN key is pressed, the receiving string variable (A$) correctly holds the entered data. If more than fifteen characters are entered before RETURN is pressed, the window scrolls and the additional characters are displayed in the input window. They are not returned into A$, however, because they don't fit in the fifteen character dimensioned length of A$. Note also the effect of the editing keys. Their major effect is on the display; no characters are removed from the string returned to A$, although some are removed from the display. The ASCII characters of the editing keys are returned, however, and they may be used in a "smart" input routine to re-construct the correct input value.

The following is a such a "smart" input routine. It is set up to input a device and filename, such as "D2:FILENAME.EXT". The maximum number of characters to be input is fifteen. The routine uses the GET function to accept characters from the window and properly responds to the use of the two editing functions. Type in the following lines

```
 10 GRAPHICS 8:MAX=15:DIM IN$(MAX+1)
 20 OPEN #1,12,1,"W:"
 30 XIO 100,#1,0,0,"W:":X=96:Y=72:POINT #1,X,Y
 40 PRINT #1;" Device & name:"
 50 X=96:Y=80:POINT #1,X,Y
 60 XIO 100,#1,1,0,"W:":X=8*(MAX+1):Y=8:POINT #1,X,Y
 70 XIO 100,#1,6,0,"W:":X=ASC("_"):Y=0:POINT #1,X,Y
 80 TRAP 170
 90 I=1:IN$=""
100 GET #1,X
110 IF X=125 THEN 90
120 IF X=126 AND I=1 THEN 100
130 IF X=126 AND I>1 THEN I=I-1:IN$(I)="":GOTO 100
140 IF X=155 THEN 170
150 IN$(I)=CHR$(X):I=I+1:IF I=MAX+2 THEN 90
160 GOTO 100
170 TRAP 40000
180 CLOSE #1
190 PRINT IN$
200 END
```

pressing RETURN after each line. Now run the program and check it out.

This piece of code can easily be modified to act as an input subroutine in any BASIC program. See also the "GETDEMO" program on your "SCREENS" diskette.

Changing the parameters

The user can change any and all of the individual window parameters at any time, providing the window is open. And likewise, the user can check on the current value of any of the parameters of any open window. The protocol for doing either operation starts by issuing an XIO command to the window of interest. The form of the XIO command for this purpose is

```
XIO 100,#iocb_no,np_index,0,device_id
```

Both iocb_no and device_id have been described previously. The variable np_index is an integer number from 0 to 10. It tells the window which parameters you wish to read or set. The table below is a partial list of allowed np_index values and the associated window parameters.

```
np_index     parameters
        0......window position (column & row)
        1......window size (width & height)
        2......character position (column & row)
        3......character cell size (width & height)
        4......character set address & size
        5......character color & display logic
        6......cursor character
        7......lock flag
```

The way the individual parameters are read and written is through the use of NOTE and POINT, respectively. For example, if you wished to set the character cell size for a given window to 13 pixels wide by 19 pixels high, you would first execute the XIO 100 command with np_index set to 3. Assuming the window was opened through IOCB 2, you would then execute the following statement

```
X=13:Y=19:POINT #2,X,Y
```

(Recall that ATARI BASIC does not allow the use of numeric constants in the arguments of the POINT statement, i.e. the statement POINT #2,13,23 would generate a syntax error.) The next character printed would fill a 13 by 27 pixel cell.

Clear the previous program, then enter a new program by typing

```
10 GRAPHICS 7
20 OPEN #1,8,1,"W:"
30 XIO 100,#1,3,0,"W:"
40 X=13:Y=19:POINT #1,X,Y
50 XIO 100,#1,5,0,"W:"
60 FOR X=1 TO 3
70 Y=0:POINT #1,X,Y
80 PRINT #1;"HELLO"
90 NEXT X
150 CLOSE #1
160 END
```

pressing RETURN after each line. Now run the program. Note that the characters in the word "HELLO" are printed in cells that are thirteen pixels wide and nineteen pixels high. Also, the three "HELLO"'s are all different colors due to the POINT statement in line 70.

When you want to check on a window parameter, use the NOTE command. For example, in the case of window size, the value of np_index would first be set to 1 by executing the XIO 100 command. Then, assuming the window was opened through IOCB 1, executing the following command

```
NOTE #1,X,Y
```

would place the column position of the upper left corner of the window in X and the row position in Y.

Add another three lines to the sample program by typing

```
100 XIO 100,#1,1,0,"W:"
130 NOTE #1,X,Y
140 PRINT X,Y
```

pressing RETURN after each line. Now run the program again. Note that the window width and height are correctly printed out as 160 and 80 respectively.

Not all the parameters are passed in pairs. For example, in the case of the cursor character only a single value is passed. In such cases, only the first parameter in the POINT command has meaning, the second parameter is arbitrary, and the NOTE command will always return zero as the value of the second parameter.

The value of np_index is readable via the STATUS statement. Assuming a window was opened through IOCB 4, you could read the current value of np_index for that window by executing the statement

```
STATUS #4,X
```

which would place the current value of np_index into the variable X.

Modify the sample program by typing

```
110 STATUS #1,N
120 PRINT N,
```

pressing RETURN after each line. Run the program again. The value of np_index is displayed along with the window width and height.

Acceptable parameter values

ATARI BASIC restricts the first parameter passed in the POINT statement to be an integer from 0 to 32767 inclusive; the first value returned by NOTE can be an integer from 0 to 65535 inclusive. The second argument of both statements must be an integer from 0 to 255 inclusive. Any value within these limits can be passed to the device "W:" without generating an error. However, if the values passed are out of range for a given parameter they may be modified or ignored.

The maximum values for window position and size depend on the display mode resolution. In turn, the maximum values for character position and size depend on window size (note: maximum character width is 255 regardless of window size). Before a character is printed to the display, these values are checked. If the column/row position equals or exceeds the number of columns/rows supported, then that parameter is set to zero. If either size parameter exceeds the maximum allowable value, then it is set to that maximum. If any size parameter is set to zero by the user, it will be set to one by the handler.

The character base pointer can be any address from 0 to 65535; it does not have to be on any particular memory boundary. Because of the above-noted limitation in the POINT statement, there is a problem in passing values greater than 32767. To get around this problem, the second argument of the POINT statement is employed. The most significant bit in the second argument is ORed with the most significant bit of the high order byte of the passed address. Thus, in order to pass a character base address greater than 32767, use the following bit of BASIC code

```
X=ADDRESS:Y=SIZE
IF X>32767 THEN X=X-32768:Y=Y+128
POINT #1,X,Y
```

Since only the least significant bit of the second argument is used to indicate the character set size, the addition of 128 will not alter its effect.

Character color can be set to any number from 0 to 255 inclusive. If a larger number is passed, only the least significant byte is kept. All eight bits of this byte are stored, but when a character is displayed only those bits needed to specify color in the active graphics mode are used. For example, in graphics mode eight, only the least significant bit controls the color; in graphics mode seven, the two least significant bits are used; and so on.

The valid values for character display logic range from 0 to 3 inclusive; a larger value may be passed, but only the two least significant bits are kept. The table below shows the logic type associated with each legal value

value	logic
0......	overwrite
1......	and
2......	or
3......	exclusive-or

Overwrite means that the bit map of the character cell to be displayed replaces the data previously present on that segment of the display. AND means that the bit map of the character cell to be displayed is logically ANDed with the current display data on a a bit by bit basis, and the result displayed. Thus, a bit will be set only if it is on in both the character cell and the previous display. The OR and EXCLUSIVE-OR options behave like the AND option except the logic used to create the actual display is different. With OR, a bit is set if it is on in either the character cell or the previous display. With EXCLUSIVE-OR, a bit in the display is unchanged if the corresponding bit in the character cell is zero and inverted if the corresponding bit is one. The exclusive-or operation is reversible; writing the same character twice to the same location using exclusive-or logic results in a return to the original display.

Attractive text displays can be created by mixing and overlaying text using different display logic. To view some, load and run the BASIC program named "LOGIC" from your "SCREENS" diskette. This BASIC program demonstrates just a few possibilities; you can explore others by modifying the DATA statements.

The cursor character can take on any value from 0 to 255 inclusive. If a larger value is passed, only the least significant byte is kept. A value of 155 for the cursor character indicates that no cursor is to be displayed.

The window lock flag is either 0 or 1; passing any value other than zero causes the parameter to be set to one, i.e. the window is locked. Locking the window has three distinct effects: (1) the window will not clear; printing an ASCII 125 to the window will cause the ATASCII symbol for this character to be displayed, (2) the window will not scroll; attempting to print past the bottom of the window will cause the text to wrap around on the bottom line and (3) the frame drawn with the XIO 102 command (see below) will invert the data under the frame instead of writing over it.

Additional NOTE/POINT functions

There are three additional NOTE/POINT functions beyond those described in the previous two sections. These are treated separately because they do not deal with window characteristics. Rather, they are used to store and retrieve window images to and from memory. The table below gives the three additional values of np_index and the parameters associated with the NOTE and POINT commands when these values are in effect.

np_index	NOTE/POINT parameters
8......	byte count/store address
9......	as above with compression
10......	start address/retrieve address

When np_index is set to 8, the NOTE function will return as its first argument the amount of memory space required in bytes to store the image of the window in memory; zero is always returned for the second argument. If the user wishes to store away the window, sufficient memory must be allocated somewhere in RAM. This can be done by dimensioning a string variable of sufficient size to hold the image or by employing unused RAM. To store away the window, the user now executes a POINT command (with np_index still equal to 8) having its first argument equal to the starting address of the allocated RAM. For memory addresses greater than 32767, the user must employ the second argument as described in the paragraph above which discussed the character base pointer.

The use of NOTE and POINT when np_index is set to nine is identical to that described in the previous paragraph. The difference in behavior is that when np_index equals nine, the window image is stored in a compressed format. Most often this will result in a significantly smaller RAM requirement to store the window. Sometimes, however, when a window image is sufficiently complex, the attempt at compression will actually require more space. The user should routinely check both methods of storage, and select the one which requires the lesser RAM space.

When a window is stored in RAM, it is not automatically erased from the display. In fact, there is no modification of the display. By storing a window away, however, the user can now modify the display at will knowing that the image has been stored away and can be recalled intact later. Once the window image is transferred to RAM, it can even be stored on cassette or disk. Other programs can recall that window image into RAM and then transfer the image to the display.

The window image is retrieved from RAM using the POINT command with np_index set to 10. All the user must do is set the first argument equal to the RAM address at which the window resides. Once again, if the address is greater than 32767, the second argument is used to indicate that condition. The same command is used whether the image was stored with or without compression. The "W:" handler is able to distinguish the two types of storage and reconstruct the window image from either.

Note that when a window is retrieved, it does not have to be retrieved to the same IOCB or to the same device unit, i.e. a window stored from "W3:" does not have to be retrieved to "W3:". Further, the window to which the image is retrieved need not be the same size or use the same graphics mode as the window from which the image was stored. When a window image is stored, information about its size is stored along with it. When the image is retrieved it will be automatically sized to fit the current window and the image pixels will be modified, if necessary, to be compatible with the current graphics mode. This is a very powerful feature of "SCREENS" which can be used to reduce or expand images and combine images from multiple sources.

The logic used when retrieving a window image to the display is determined by the same parameter which determines the character display logic. It is set, as described previously, by using the XIO 100 command with np_index set to 5. The display logic values are interpreted the same for window retrieval as for character display. Thus, a retrieved window image can overwrite or logically AND, OR or EXCLUSIVE-OR with the current display data.

The last function to be discussed is the use of the NOTE command with np_index set to 10. In this case, the value returned for the first parameter is the starting address of the "W:" handler in RAM; zero is always returned for the second parameter. Since the handler does not need to reside at a set address, this function is a convenience for those users who need to know the handler's exact RAM location.

Other XIO commands

There are two additional XIO commands recognized by the "W:" handler besides the XIO 100 command used to set np_index. These are XIO 101 and XIO 102. Both can be used to frame the current window, but they act in somewhat different fashions.

When the XIO 101 command is issued with both parameters set to zero, a solid frame is drawn, one pixel wide, at the inside boundary of the window. The frame is actually part of the window and will be cleared, scrolled and stored along with the rest of the window. If the first parameter is non-zero then the boundaries of the window are first moved out that number of pixels before the frame is drawn. If the second parameter is non-zero then the boundaries of the window are moved in that number of pixels after the frame is drawn. Thus, assuming the window is not at the outer boundaries of the display, one could draw a frame one pixel wide just outside the current window boundaries by using the command

```
XIO 101,#1,1,1,"W:"
```

The window will grow by one pixel in all directions, the frame will be drawn inside that new boundary, then the window will be reduced to its original size.

When the XIO 102 command is issued with both parameters set to zero, all the data inside the window is inverted, i.e. all the bits which are 0 are set to 1 and all the bits that are 1 are set to 0. If the first parameter is non-zero then the boundaries of the window are first moved out that number of pixels before the window is inverted. If the second parameter is non-zero then the boundaries of the window are moved in that number of pixels after the frame is inverted, and then the window is inverted again inside its new boundaries. Thus, assuming the window is not at the outer boundaries of the display, one could draw a frame one pixel wide just outside the current window boundaries by using the command

```
                    XIO 102,#1,1,1,"W:"
```

The window will grow by one pixel in all directions, the image will be inverted inside that boundary, then the window will be reduced to its original size and inverted again.

Note that there is a distinct difference between drawing the frame this way using XIO 102 as opposed to employing the XIO 101 command. Using XIO 102, you have not destroyed the image which previously resided in the area which is now "frame" since it has just been inverted and not overwritten. (Note: this is also true for the XIO 101 command if the window is locked.) The process can be reversed by repeating the same XIO 102 command used to draw the frame, and the display is now returned to its original condition. Furthermore, the size of the frame drawn by the XIO 102 command is a function of the size of the arguments. To draw a frame four pixels wide, use the command

```
                    XIO 102,#1,4,4,"W:"
```

The command

```
                    XIO 101,#1,4,4,"W:"
```

on the other hand, will draw a single pixel frame spaced for pixels away from the edges of the original window.

As with all XIO commands, the values of the two parameters are limited to integers from 0 through 255 inclusive.

Handler errors

The only error generated by the "W:" device handler is ERROR 146. This error will occur in two situations: (1) an attempt to use an unsupported XIO command, e.g. XIO 122,#1,0,0,"W:", or (2) an attempt to set np_index to a value greater than ten, e.g. XIO 100,#1,43,0,"W:". The handler will "pass along" the ERROR 136 which is generated if a control-3 is entered while inputting text from "W:". Also, if the BREAK key is pressed during input or output, the program will be STOPPED.

character color	the color used to display each character. Set/read as the first argument of the POINT/NOTE command executed with np_index set to five.
character display logic	determines how a character or retrieved window merges with data on a display. Set/read as the second argument of the POINT/NOTE command executed with np_index set to five. value logic 0......overwrite 1......and 2......or 3......exclusive-or

character font	bit map of the character set to be displayed. The starting address of the character font is set/read as the first argument of the POINT/NOTE command executed with np_index set to four. If the address is greater than 32767, it can be passed with the POINT command by using the following BASIC code
	`IF X>32767 THEN X=X-32768:Y=Y+128`
	`POINT #1,X,Y`

character position	the offset in pixels from the upper left corner of the window to the upper left corner of the next character cell to be filled. The horizontal and vertical offsets are set/read as the first and second arguments of the POINT/NOTE command executed with np_index set to two.
character size	the size in pixels of the character cell. The height and width are set/read as the first and second arguments of the POINT/NOTE command executed with np_index set to three.

clearing a window	is performed by outputting an ASCII 125 in a PRINT or PUT command, e.g. PUT #1,125. When a window is cleared, the window area is blanked (all display data set to zero) and the character position is automatically set to zero horizontal and vertical offset. If a window is locked, the window will not be cleared and the "SHIFT CLEAR" symbol will be displayed.
closing a window	is performed with the CLOSE command, e.g. CLOSE #1. Neither the window parameters nor the display is effected.
cursor character	is the character displayed as a cursor when inputting from a window. The ASCII value of this character is set/read as the first argument of the POINT/NOTE command executed with np_index set to six. If its value is 155, then no cursor is displayed.
font size	is set/read as the second argument of the POINT/NOTE command executed with np_index set to four. A value of 0 indicates an 8 by 8 character set, any other value indicates a 16 by 16 character set.
framing a window	is performed using either the XIO 101 or XIO 102 command, e.g. XIO 101,#1,0,0,"W:" could be used to draw a frame just inside the current window boundary. See the section on "Other XIO commands" for variations.
handler location	is read as the first argument of the NOTE command executed with np_index set to 10. Its value is the starting address of the RAM memory occupied by the "W:" handler.
inputting text from a window	is performed using the GET or INPUT statement, e.g. GET #1,X. The data returned are the ASCII values of the keyboard input. As the data is input, it is displayed using the currently defined character set, size, color and logic.

inverting a window is performed using the XIO 102 command, e.g. XIO 102,#1,0,0,"W:". All the bits in the window are inverted, i.e. all zeroes become ones, all ones become zeroes.

loading a window image from disk is performed in a three step process. First, the initial two bytes of the image file must be read to determine the window image size. Then a space in memory must be located of sufficient size to hold the data. Finally, the remainder of the data is read from the file into memory. The following BASIC code demonstrates this process.

```
OPEN #1,4,0,"D:FILENAME.WOW"
GET #1,L:GET #1,H:SIZE=256*H+L
DIM IMAGE$(SIZE)
IMAGE$(1)=CHR$(L):IMAGE$(2)=CHR$(H)
FOR I=3 TO SIZE:GET #1,X
IMAGE$(I)=CHR$(X):NEXT I
CLOSE #1
```

loading window parameters from disk assumes the data has been stored in a file as described under "saving window parameters to disk". The following BASIC code demonstrates the process.

```
OPEN #1,12,1,"W:"
OPEN #2,4,0,"D:FILENAME.PRM"
FOR I=0 TO 7
XIO 100,#1,1,0,"W:"
INPUT #2,X:INPUT #2,Y
IF X>32767 THEN X=X-32768:Y=Y+128
POINT #1,X,Y:NEXT I
CLOSE #1:CLOSE #2
```

locked & unlocked window status	is determined by the state of the window lock flag. This flag is set/read as the first argument of the POINT/NOTE command executed with np_index set to seven. A value of 0 indicates an unlocked window; any other value indicates a locked window.
np_index	is set as the first argument of the XIO 100 command, e.g. XIO 100,#1,7,0,"W:". Its value can be read using the STATUS command, e.g. STATUS #1,X. Valid values for np_index are zero through ten inclusive.
opening a window	is accomplished using the OPEN command, e.g. OPEN #1,8,1,"W:". The window parameters are reset if the third argument is non-zero, as it is in the example, otherwise they are unchanged.
overlaying windows	can be accomplished by (1) opening the window to overlay the current display and defining its parameters, (2) storing the current window contents to RAM, then (3) clearing the window. The window can now be printed to, input from, etc. When the use of the overlay window is complete, the previously stored window image is retrieved from RAM with overwrite display logic and the overlay window closed.
printing text to a window	is performed using the PUT or PRINT statement, e.g. PUT #1,X. The data output to "W:" are the ASCII values of the characters to be displayed. They are displayed using the currently defined character set, size, color and logic. Two special characters are recognized: ASCII 125 which clears the window and ASCII 155 which causes a display "carriage return/line feed", moving the character position to the beginning of the following line.

retrieving a window from RAM is accomplished using the POINT command with np_index set to ten. The first argument is the address in RAM of the window image data. The following BASIC code is an example of this process.

```
X=ADR(IMAGE$):Y=0
IF X>32767 THEN X=X-32768:Y=Y+128
POINT #1,X,Y
```

saving a window image to disk is accomplished by saving to disk that part of RAM into which the window image has been stored.

```
OPEN #1,8,0,"D:FILENAME.WOW"
PRINT #1;IMAGE$;
CLOSE #1
```

See "storing a window image to RAM".

saving window parameters to disk is accomplished by reading the parameters using the NOTE function and writing them to a file. The following BASIC code demonstrates the process.

```
OPEN #1,12,0,"W:"
OPEN #2,8,0,"D:FILENAME.PRM"
FOR I=0 TO 7
XIO 100,#1,I,0,"W:"
NOTE #1,X,Y
PRINT #2;X:PRINT #2;Y
NEXT I
CLOSE #1:CLOSE #2
```

storing a window image to RAM — is performed in a three step process. First, the window image size is determined by using the NOTE command with np_index set to eight (normal) or nine (compressed). Then a space in memory is reserved of sufficient size to hold the data. Finally, the window image is transferred to RAM using the POINT command with np_index set to eight or nine. The following BASIC code demonstrates this process.

```
XIO 100,#1,8,0,"W:"
NOTE #1,NSIZE,DUMMY
XIO 100,#1,9,0,"W:"
NOTE #1,CSIZE,DUMMY
SIZE=CSIZE:TYPE=9
IF NSIZE<SIZE THEN SIZE=NSIZE:TYPE=8
XIO 100,#1,TYPE,0,"W:"
DIM IMAGE$(SIZE)
X=ADR(IMAGE$):Y=0
IF X>32767 THEN X=X-32768:Y=Y+128
POINT #1,X,Y
```

window position — the offset in pixels from the upper left corner of the display to the upper left corner of the window. The horizontal and vertical offsets are set/read as the first and second arguments of the POINT/NOTE command executed with np_index set to zero.

window size — the size in pixels of the window. The height and width are set/read as the first and second arguments of the POINT/NOTE command executed with np_index set to one.

Introduction

The purpose of this section is to provide some additional details on the structure and workings of the "W:" device handler. No additional features will be described. The information will be most useful to those interested in non-BASIC applications or applications which require the handler to be co-resident with other software. This section is not meant to be tutorial in nature; many sections assume an intimate knowledge of the ATARI operating system.

Program installation

The "W:" device handler resides in a 33 sector AUTORUN.SYS file. This file loads into RAM at starting address $6C00 (the 27KB boundary). It initially occupies just under 4KB of RAM space. The starting address was chosen so that, in a multiple file boot, "SCREENS" would load beyond any non-relocatable application with which it was merged, yet be compatible with a 32kB system.

The application is installed by an INIT routine which is later jettisoned. This routine's primary functions are to (1) load the 850 interface module handler if the 850 is connected and powered up, (2) relocate the "W:" device handler so that it begins at the current MEMLO address, (3) adjust all the absolute memory addresses in the handler code to adapt to its new location and (4) point the DOSINI vector to the handler initialization routine. The handler initialization routine, which after installation is vectored to on SYSTEM RESET, first does a JSR to the vector which hac been in DOSINI just prior to installation of the "W:" handler. It then re-installs the device in the handler table, repositions MEMLO to just beyond the device handler code and resets the parameters for all nine device units.

The application also has a RUN vector which points to a routine which puts the program title, the version number and the copyright notice on the display. This RUN routine is not critical to handler operation. Thus, the "W:" handler AUTORUN.SYS file need not be the final file in a compound AUTORUN.SYS file, i.e. its RUN routine does not need precedence. Although some thought has been given to compatibility with other AUTORUN programs, certainly there are cases which will not yield to a simple merging of the files. In particular, any file which does not relocate yet has a RUN routine which must be executed cannot be easily merged.

Memory usage

Once it is installed, the "W:" device handler uses only two blocks of RAM. The largest block, of course, is the 3297 bytes occupied by the handler itself. Because the handler is relocatable, the position of the block is not fixed. For this reason, the NOTE function with np_index set to ten was designed to provide the starting address of the handler routine.

The other block of employed memory is on page zero. Thirty-two bytes from $E0 to $FF inclusive are used as working space whenever the "W:" device is accessed. Other routines can use this zero-page memory with impunity, however, because the "W:" handler puts the values found there in temporary storage while it's working, then restores them when it has completed its job. The only necessary precaution is with interrupt routines; if an interrupt routine employs any of the handler's zero page locations, it must do so in a fashion which restores them to their original value when the interrupt is complete.

OS variables used

The "W:" handler relies on the following operating system variables to be used as described in the OS Technical Users' Notes: BOTSCR, BRKKEY, DINDEX, DOSINI, HATABS, ICAX1Z, ICAX2Z, ICAX3, ICAX4, ICAX5, ICCOMT, ICCOMZ, ICDNO, ICDNOZ, MEMLO, SAVMSC, SIOCB, SIOV, SSFLAG. Some of these variables rate individual attention.

SAVMSC is assumed to point to the first byte of the memory mapped display. BOTSCR and DINDEX together are used to describe the current graphics display mode; they determine the display and pixel size used by the "W:" handler. Note that the handler does not recognize mixed mode displays other than the standard graphics mode with text windows. Using "W:" in a custom mixed mode display is possible, but tricky, since only a single resolution and display size is assumed. Manipulation of SAVMSC and DINDEX will give you some degree of control.

ICCOMZ contains the function number of "special" commands. This applies to the XIO function number. This also applies to the "special" commands NOTE and POINT. The NOTE command uses the function number $26; the POINT command uses the function number $25.

ICAX1Z and ICAX2Z are where CIO puts the values passed in the first and second arguments respectively of the OPEN and XIO commands. The values passed with the NOT and POINT commands are placed in locations ICAX3, ICAX4 and ICAX5. In particular, ICAX3 and ICAX4 contain the low and high byte respectively of the first argument; ICAX5 contains the value of the second argument.

One point already mentioned numerous times is the limitation ATARI BASIC puts on the value which may be passed as the first argument of an XIO command. This necessitated some additional code when the value exceeded 32767. The way the handler determines the value of the high byte of the first argument passed by an XIO command is using the following code

```
                    LDA ICAX5,X
                    AND #$80
                    ORA ICAX4,X
```

The most significant bit of the first argument (ICAX5) may effect the high byte of the first argument (ICAX4).

Window image format

The window image is comprised of two parts - an eight byte header and the actual image data. The first two bytes of the header are the low and high bytes respectively of the count of the total number of bytes in the window image, including header. The next two bytes of the header are the low and high bytes respectively of the number of pixel columns in the window. The next two bytes of the header are the low and high bytes respectively of the number of pixel rows in the window. The next byte of the header is the number bits per pixel in the window; sixteen is added to this number if the window data is in compressed format. The final byte of the header is a mask byte showing where the valid window data begins in the first image data byte. This mask is necessary because the window image is not "left-justified" inside the image data, i.e. it does not necessarily begin at a byte boundary. Thus, it is necessary to indicate where the valid image data begins. This is done by the bits which are set in the mask byte.

In the uncompressed storage format, all bytes in a line which hold any part of the window image are stored in order from left to right. Image data is stored in a line by line format, top to bottom, just as it appears in the display RAM. This is akin to the "standard" method of storing an ATARI graphics image. In fact, appending the appropriate header to a "standard" graphics image file will convert it into a window image file.

Let's take an example of storing a window image which is in fact a whole GRAPHICS 24 display. What would the header for this image look like if it were stored uncompressed? The header bytes would have the following values:

8 30 64 1 192 0 1 255

There are 8+30*256=7688 total bytes in the image. The image is 64+1*256=320 pixels wide by 192+0*256=192 pixels high. There is 1 bit per pixel. All the data in the first byte is valid (all bits of the mask are 1).

In the compressed storage format, the same data bytes are stored and in the same order as described above. However, the data is stored in sequences. Each sequence is comprised of a count byte followed by one or more data bytes. There are two types of sequences. If the count byte has a value of 0 to 127 inclusive, then it is a repetitive sequence representing the repeat of the following data byte count+1 times. If the count byte is 128 to 255 inclusive, then it is a non-repetitive sequence of count-127 distinct data bytes.

Using this technique to store the data pattern {1 2 3 4 4 4 4 4 4 4 4 6 6 6 6 6 6 6 6 6 6 7 8 9} would result in the data pattern {130 1 2 3 7 4 9 6 130 7 8 9}. The example shows a compression of 50%, from 24 to 12 bytes. Obviously, the degree of compression depends strongly on the repetitiveness of the data and, in fact, the compressed mode can sometimes require more storage space than the uncompressed mode. For that reason, it is always prudent to check both means of compression to determine the more efficient one.

Expanded font format

The "W:" handler will accept both 8 by 8 and 16 by 16 character fonts. The format of the former has been established by the format of the standard ATARI character font in ROM. Each character is represented by eight bytes; each byte represents one line in the character in sequence from top to bottom. Each bit in a byte represents a pixel in the 8 by 8 character cell; if it's set then that bit is "on" when the character is displayed. The 16 by 16 character set is laid out in an analogous fashion. Each character of the set is represented by 32 bytes of data. Each pair of bytes represents a line of the character in sequence from top to bottom. The first byte of each pair represents the eight leftmost bits of the character cell; the second byte, the rightmost bits. The order of display of the bits is the same as in the 8 by 8 set, namely, the most significant bit in the byte is the leftmost pixel. The order of the characters in the 16 by 16 character set is identical to that in the 8 by 8 set.

A BASIC program for creating and editing 16 by 16 character sets also resides on your "SCREENS" diskette. For documentation, RUN "D:FONT16.DOC".

Useful tables

XIO number	function
100	set np_index
101	frame
102	invert

np_index	NOTE/POINT parameters
0	window position (column & row)
1	window size (width & height)
2	character position (column & row)
3	character cell size (width & height)
4	character set address & size
5	character color & display logic
6	cursor character
7	lock flag
8	byte count/store address
9	as above with compression
10	start address/retrieve address

logic parameter value	logic type
0	overwrite
1	and
2	or
3	exclusive-or

```
   default window characteristics
window position: column=0, row=0
window size: width=320, height=192
character position: column=0, row=0
character cell size:  width=8, height=8
character set: base address=57344 (ROM),
               matrix=8x8, color=255
               display logic=overwrite
cursor character=160 (inverse space)
lock flag=0 (unlocked)
np_index=0
```